

Stored Procedure/Function Error Handling (PostgreSQL)

Stored procedures in PostgreSQL can contain many different types of SQL statements including queries, Inserts, Updates, And Deletes. A DML (Data Manipulation Language) statement inside a stored procedure works exactly as it does outside of a stored procedure. However, there are some additional considerations when using DML statements inside stored procedures, especially when they are called from client applications (for example, Java, Python, or .NET). If an error occurs (such as a constraint violation or invalid data), PostgreSQL returns a detailed error message to the client. These messages are often technical and may not be meaningful to the end user. To make error handling more user-friendly, we can intercept any errors in our procedure and raise a clearer message.

Functions can include robust error handling even when they do not modify data. This is especially useful when performing calculations, data lookups, or conversions where invalid inputs or runtime issues may occur.

To handle errors gracefully in PostgreSQL, we use a Begin ... Exception ... End block. This structure lets us catch errors and raise more meaningful messages. When an error is handled in an Exception block, the PL/pgSQL local variables keep the values they held when the error happened, but any changes made to the database within that block are rolled back.

Simplified Syntax:

Begin

 -- Code that may cause an error

Exception

 When condition Then

 -- Handle the error

End;

Examples of handling possible exceptions:

Begin

-- Code that may cause an error

Exception

When too_many_rows Then

Raise Exception 'Duplicate record found error: %', SQLERRM;

When no_data_found Then

Raise Exception 'Record not found error: %', SQLERRM;

When unique_violation Then

Raise Exception 'Key value not unique error: %', SQLERRM;

When others Then

Raise Exception 'Unknown error: %', SQLERRM;

End;

Notes:

- SQLERRM contains the error message text (not usable outside exceptions)
- When “others” catches all remaining exceptions.
- Raise Notice – gives warning message to client (i.e. pgAdmin)
- Raise Exception – raises an error to client (i.e. pgAdmin) and stops processing

Some common integrity exception examples are not_null_violation, foreign_key_violation, unique_violation, check_violation.

Some common data exception examples are division_by_zero, datetime_field_overflow, invalid_datetime_format, numeric_value_out_of_range.

Create a stored procedure that inserts a new club record. If an error occurs such as trying to insert a duplicate primary key, raise a clear error message.

Create Procedure PR_Add_Club

(p_ClubID varchar(10), p_ClubName varchar(50))

Language plpgsql

As \$\$

Begin

Insert Into Club

(ClubID, ClubName)

Values

(p_ClubID, p_ClubName);

Raise Notice 'Club % added successfully', p_ClubID;

Exception

When unique_violation Then

Raise Exception 'A club with ID % already exists', p_ClubID;

When others Then

Raise Exception 'An unexpected error occurred inserting the club
% Error: %', p_ClubID, SQLERRM;

End;

\$\$;

To run:

Call PR_Add_Club ('NewID', 'New Club Name');

Create a stored function that returns a student's ID when passed in a first name. If an error occurs, such as a duplicate or non-existent first name, raise clear error messages. You must use 'Select...Into Strict...' to ensure the exceptions fire if the first name is not unique or does not exist. Without 'Select...Into Strict...', the database would populate the local variable with the first occurrence if there were multiple first names, or null if there was a non-existent first name.

Create Function FN_Get_Staff_ID

(p_FirstName varchar (50))

Returns integer

Language plpgsql

As \$body\$

Declare

v_StaffID integer;

Begin

Select StaffID

Into Strict v_StaffID

From Staff

Where FirstName = p_FirstName;

Raise Notice 'Unique staff first name % exists', p_FirstName;

Return v_StaffID;

Exception

When No_Data_Found Then

Raise Exception 'Staff first name % not found', p_FirstName;

When Too_Many_Rows Then

Raise Exception 'Staff first name % is not unique', p_FirstName;

When Others Then

Raise Exception 'Unknown error: %', SQLERRM;

End;

\$body\$

To run:

```
Select FN_Get_Staff_ID ('Cher');
```

Optional: recreate the Function with a default value for first name:

```
Drop Function FN_Get_Staff_ID (p_FirstName varchar (50));
```

```
Create Function FN_Get_Staff_ID
```

```
(p_FirstName varchar (50) = null)
```

```
Returns integer
```

```
Language plpgsql
```

```
As $body$
```

```
Declare
```

```
    v_StaffID integer;
```

```
Begin
```

```
    If p_FirstName Is Null Then
```

```
        Raise Exception 'You must enter a staff first name';
```

```
    Else
```

```
        Begin
```

```
            Select StaffID
```

```
            Into Strict v_StaffID
```

```
            From Staff
```

```
            Where FirstName = p_FirstName;
```

```
            Raise Notice 'Unique staff first name % exists', p_FirstName;
```

```
            Return v_StaffID;
```

```
        Exception
```

```
            When No_Data_Found Then
```

```
                Raise Exception 'Staff first name % not found', p_FirstName;
```

```
            When Too_Many_Rows Then
```

```
                Raise Exception 'Staff first name % is not unique', p_FirstName;
```

```
            When Others Then
```

```
                Raise Exception 'Unknown error: %', SQLERRM;
```

```
        End;
```

```
    End If;
```

```
End;
```

```
$body$
```

To run:

```
Select FN_Get_Staff_ID ('Cher');
```

```
Select FN_Get_Staff_ID ('Bob');
```

```
Select FN_Get_Staff_ID ();
```